Tampere University of Applied Sciences

Analysis and Comparison of Unity and FMOD Sound Engines

Akseli Takanen

BACHELOR'S THESIS
May 2020

Degree programme in Media and Arts
Music Production

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree programme of Media and Arts
Music Production

AKSELI TAKANEN:
Analysis and Comparison of Unity and FMOD Sound Engines

Bachelor's thesis 37 pages
May 2020

This study compared Unity game engine's built-in audio engine with third-party audio engine FMOD. The analysis was theoretical based on literary and educational sources as well as practical project examples provided by Skydome Entertainment Oy.

The study began with a rundown of the technical features of both audio engines. Functions such as audio file management, compressing and loading assets, audio sources and listeners, mixing, routing, sound shaping and performance profiling were examined. The following comparison focused on few key aspects of the audio engines: technical features, workflow, ease of use and resource investment. The main research question was which of these popular audio engines has the higher potential for creating a quality game audio system with the most resource efficiency. The analysis was made from the standpoint of sound design potential, flexibility and resource costs including time efficiency of the sound designer as well as programmers, and financial expenses.

The results of the analysis were in favor of FMOD as the technically superior and potentially more efficient audio engine. However, the challenges regarding the level of audio engineering and sound design proficiency required to operate the engine to its full capacity were also brought up. Additionally, certain scenarios in which the Unity audio engine might be the better choice were discussed.

The final segment of the study touches on the long-term potential and future of the two audio engines. There is room for more methodical studies especially regarding the efficiency of the audio engines which could benefit game companies in streamlining their production workflow while being able to create high quality audiovisual systems.

Key words: unity, fmod, game development, sound design

CONTENT

1 INTRODUCTION

Sound design in video games is a field that requires specific tools and expertise in addition to audio engineering skills. Differing from the linear nature of television and film as media, games from technical point of view are mostly nonlinear – The player's actions are unpredictable as opposed to a film, which always plays the same from start to finish. Major aspect of video game sound design is creating a complete sound system, which reacts to the player's actions, provides necessary feedback from those actions and ultimately contributes to the artistic quality of the game.

In order to create a sound system, the sound designer must know how to operate an audio engine, which is integrated into the game engine. Research, forethought and smart choices create a solid foundation for the sound system which in the long run saves time and resources of the whole development team. This study analyses and compares the two most common audio engines for small to mid-sized game projects: Unity's built-in audio engine and a third-party audio engine FMOD Studio. Through this analysis the purpose is to give game developers and sound designers a better understanding on how these sound engines differ in features, workflow, skill requirements and resource cost. Ultimately, the goal is to help game developers to better plan and streamline audio work and get more satisfactory results while minimizing any unnecessary workload for sound designers and programmers.

The study is made in co-operation with Skydome Entertainment Oy who allow the use of picture examples from their projects. The examples showed in the study are from games The Hive (released 2016) and project Kingdom Fall, which is still in development at the time of writing. The Hive's audio system is made with Unity's built-in audio tools, while the audio implementation for Kingdom Fall is done with FMOD Studio. Both games utilize the Unity game engine.

2 INTRODUCTION TO UNITY AND FMOD

Unity by Unity Technologies is a real-time 3D development platform for game and media designers, programmers and artists. It is one of the leading and most widely used game engines in the industry. Unity Technologies CEO John Riccitiello (2018) has stated the following about the number of games powered by the engine:

> It's pretty much half of all games period. We have different market shares depending on the platform. But more than half of all mobile games built for there are built in Unity. (Dillet, 2018.)

Unity does offer tools for all aspects of game production – including sound design. However, it is reasonable to examine and compare its capabilities to third-party options.

In the field of audio implementation FMOD is one of the industry standards along with Wwise developed by Audiokinetic. Similar to FMOD Wwise offers extensive sound design tools for creating a high-quality sound system. This study, however, focuses on FMOD for couple of reasons. While both software offers in-depth audio implementation features, FMOD is possibly slightly easier to approach due to its simplistic and efficient visual presentation as well as better documentation. Furthermore, the indie level FMOD license is free for game projects with a budget under $500k with no limitations (FMOD Licencing, 2020). The free Wwise licence on the other hand is limited to only 500 audio assets (Audiokinetic Pricing, 2020). While prices of the lower tier Wwise licences are realistic for small indie projects as well, FMOD is a very accessible option for projects of all budgets and scopes.

In order to understand and compare the two audio engines it's beneficial to do a basic overview of their respective core features. Some of the features examined are:

- Importing assets
- Compressing and loading audio clips
- Audio listener and audio sources
- Mixer and audio hierarchy

- Effects and real-time audio processing
- Profiler and performance metering

The main focus of the analysis is in the most commonly used features and tools in game sound design. The version of Unity used is 2019.3, which is the latest stable version at the time of writing. For FMOD the version used is 2.00.03, which is integrated to Unity 2019.3.

2.1 Overview of Unity's audio features

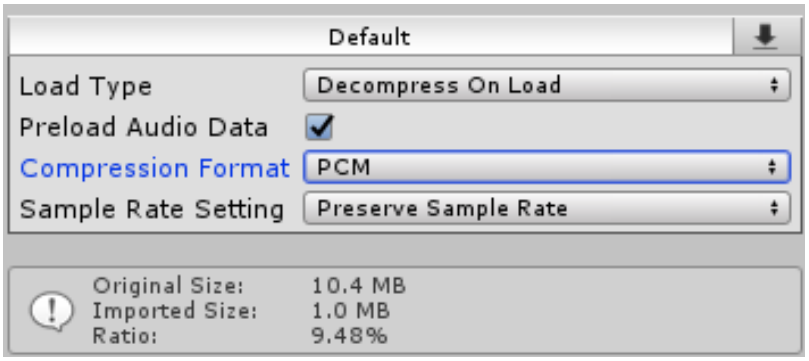
Unity 3D by Unity Technologies is the development platform used in this analysis due to it being the most widely used game engine in the industry at the moment. The analysis specifically focuses on Unity's 3D functions and sound design in its context.

Unity's audio system features support for 3D spatial sound, mixer hierarchies and real-time audio processing among other features. Unity does also support Ambisonic audio for VR and 360° video projects. However, Ambisonic audio clips bypass the standard audio pipeline so an Ambisonic Decoder plug-in must be installed first in order to hear and be able to process them. (Unity Documentation, 2019.) Nonetheless, this study focuses on the standard audio pipeline in more typical sound design scenarios.

2.1.1 File formats and Load Types

Unity supports most standard audio formats like WAV, MP3, OGG and AIFF. There are three compression formats used for sounds at runtime. PCM is a high-quality format, although large in file size. It's only recommended for short sound effects when loss of sound quality is not desirable. ADPCM is a lower quality format, which is convenient for sounds played in large quantities. The third option is Vorbis/MP3, which could be considered a middle option between the three. For

Vorbis/MP3 there is a quality setting, which determines the amount of compression. (Unity 3D Documentation, 2019.) In addition, Unity supports MOD and XM among other tracker formats (Hocking, 2018).



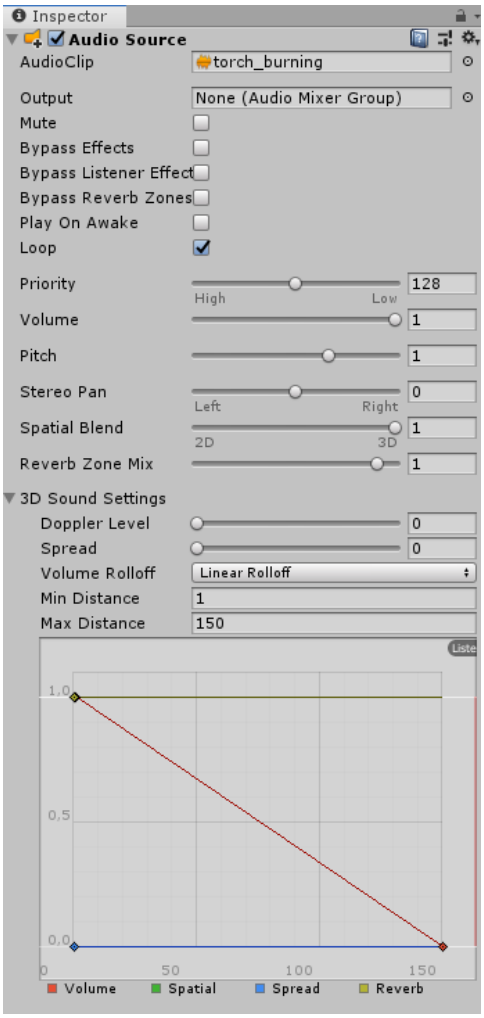
PICTURE 1. Compression Formats and Load Types in Unity (Takanen A. 2020.)

There are three options available for loading audio assets at runtime. The use of a load type depends on the length and purpose of the audio clip as well as desired memory use. *"Compressed in Memory"* means that the audio clip is stored in RAM and uncompressed when played in the scene. Playing doesn't require any additional memory. *"Streaming"* option means that the audio clip is stored on a device persistent memory like hard drive and streamed when played. It doesn't require any significant amount of RAM for storing or playing. The third option *"Decompress on Load"* means that the audio clip is stored in RAM uncompressed. It's the heaviest option for memory but requires the least CPU. (Unity Documentation, 2019.) Different load types allow sound designer to optimize memory and CPU use which is extremely important especially on development platforms like mobile with significant performance limitations.

2.1.2 Audio Listener and Audio Sources

The Audio Listener is a component, which receives input from any audio sources in the scene and plays the sounds through the main output. By default, it's attached to the Main Camera, which is suitable for most applications (Unity Documentation, 2019). There can only be one Audio Listener in a scene. The Audio Listener doesn't have any additional properties – it's simply required to receive audio input. However, it's possible to apply effects to the Audio Listener.

The Audio Listener works in conjunction with Audio Sources of the game scene. Audio Sources can be 3D, which makes the Audio Listener emulate positioning and amplitude changes in a 3D space. 2D Audio Sources on the other hand completely ignore the 3D parameters and processing. The parameters that affect the Audio Sources behavior in the scene can be found in the Audio Source Component.



PICTURE 2. The Audio Source component (Takanen A. 2020.)

Audio Source component also has controls for sound clip specific audio behaviour like volume and pitch controls, routing, muting and bypassing effects among other functions.

2.1.3 Mixer and audio hierarchy

Unity has a built-in mixer for audio routing, grouping, mixing and effects processing. Audio Sources can be routed to specific tracks in the mixer, which can be further routed to groups and buses. The Audio Mixer is essentially taking place after the Audio Source in the signal chain allowing processing and structuring of the audio hierarchy before it is output via the Audio Listener. (Unity Documentation, 2019.) Groups within the mixer are integral for controlling different categories of sounds – for example ducking certain SFX categories when dialogue is playing.

The mixer view includes standard track controls like volume faders, mute, solo and bypass buttons. It's also possible to adjust individual audio clip's volume level in the Audio Source component. However, instead of decibel attenuation, the volume slider in the Audio Source is percentage based which makes it more difficult to adjust for sound designers who are used to decibel measurement. The Audio Mixer is a more capable tool for mixing since it has standard decibel attenuation.

In Unity multiple audio mixers can be created which is useful for projects with particularly large number of audio assets. It can also be beneficial to have separate mixers for music, SFX and maybe FX return tracks to keep the hierarchy organized. Additionally, the mixer view can be altered by turning tracks invisible when necessary to further simplify the layout (Philipp, 2015.)

It's possible to create Snapshots, which capture the state of all parameters in the mixer hierarchy. Snapshots are useful for situations where the whole mix is altered in a specific way, like in-game menus where certain categories of sounds need to be muted entirely. Another common use for Snapshots is changing area specific FX like reverbs between different zones in a game. Snapshots are perhaps one of the most powerful features in Unity's audio tool kit. They make it possible to introduce dynamics into a game's soundscape which greatly adds to the immersion and liveliness of the audio-visual presentation. However, transitioning from a Snapshot to another requires creating a script, which makes the switch happen when a set incident occurs in a game – for example one object hits another. Unity documentation provides more information about audio and Snapshot related scripting (Unity Documentation, 2020).

PICTURE 3. Mixer hierarchy in Unity (Takanen A. 2020.)

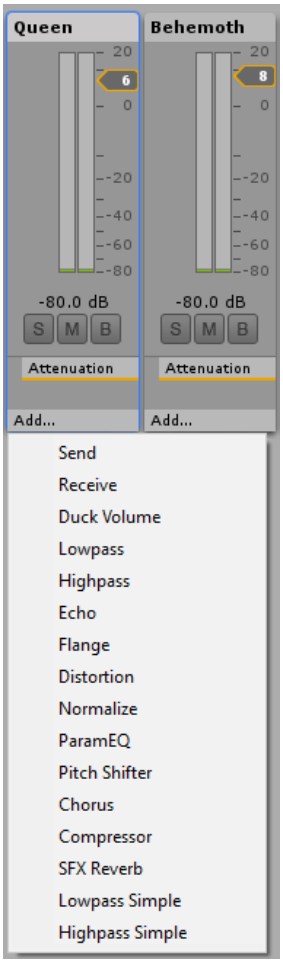
2.1.4 Audio effects

Most audio effects are accessed from the Audio Mixer. The only effect that is controlled from the Audio Source view, however, is the doppler effect. The doppler effect can be used for 3D sounds to emulate the change of frequency in relation to the observer, which in sound design context the Audio Listener often attached to the main camera.

Unity provides several commonly used effects like filters, EQ, reverb, distortion, compression and chorus. They can be used as track specific insert effects or auxiliary effects. In the effects selection also Send and Receive are listed, which are inserted to tracks whenever audio is routed to buses or auxiliary effect tracks. Send effect tracks must be made manually by first creating a track with a reverb insert effect in it and sending all desired tracks to it by adding the Send insert to them. (Fisher, 2017.)

Effect parameters can be automated with scripts. For example, the pitch of a sound instance like a footstep can be randomized by inserting the Pitch Shifter effect and setting a range of values for the pitch shift parameter to randomize on each instance of the footstep. Parameter automation, randomization and live manipulation does require certain amount of familiarity with audio scripting in Unity. Another, perhaps slightly clumsier option for pitch randomization is controlling the pitch parameter in the Audio Source itself. The benefit of using the mixer's Pitch

Shifter effect is that it can be inserted to groups or buses in order to process several Audio Sources at once with a single effect instance.



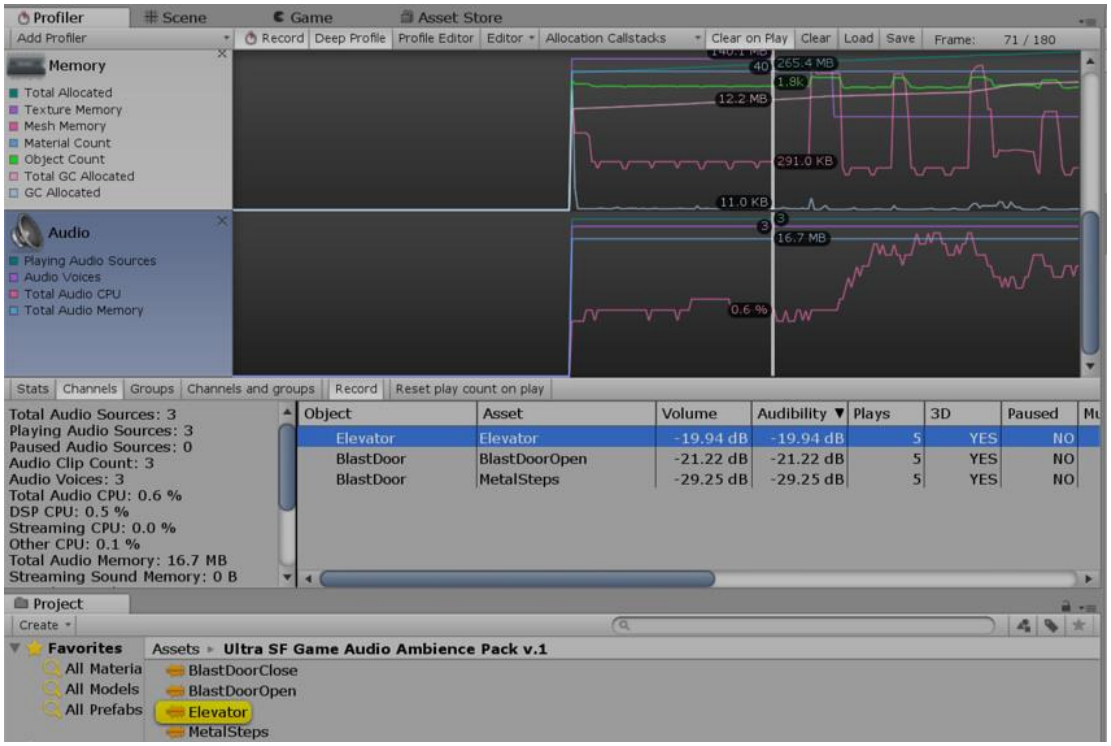
PICTURE 4. Audio Effects in the Unity Mixer (Takanen A. 2020.)

2.1.5 Audio Profiler

Unity's Profiler window makes it possible to monitor and collect data of the game's performance in real time. The Profiler has a section dedicated to getting performance information specifically about the audio engine.

Playing Audio Sources indicates how many Audio Sources are playing in the game scene at a specific frame. Total Voices, however, tells how many actual voices, like FMOD tracks, are being played, which are not shown in Playing Audio Sources. An important graph to keep eye on is Audio Memory, which indicates

how much RAM the audio engine uses in total. CPU usage of the audio can be monitored as well. (Unity documentation, 2019.)



PICTURE 5. Audio Profiler in Unity (Unity User Guide. 2020.)

The Audio Profiler also shows which audio clips are being played from which game objects, the volume they are played at and how far they are from the listener. The Audio Sources and audio clips themselves can be accessed by selecting them from the Profiler view. Overall, the Audio Profiler is extremely useful tool for troubleshooting and optimizing the audio engines performance.

2.2 Overview of FMOD's audio features

FMOD is a third-party audio middleware developed by Firelight Technologies. FMOD can integrated as a primary sound engine to several game engines including Unity.

Detailed steps for Unity integration are not covered in the analysis. However, to summarize FMOD Studio app as well as corresponding version of FMOD Unity integration package need to be installed. After the FMOD integration package is

downloaded and imported to Unity an FMOD project file needs to be selected from Unity's FMOD integration settings screen to make it the main audio editor. Finally, Unity's built-in audio engine should be disabled completely. (FMOD documentation, 2019.)

Overall, FMOD's audio features are similar to Unity's yet more expansive FMOD Studio being a dedicated sound effects engine. The drawback, however, is working with two separate programs operating together which as a setup is always more sensitive to technical issues or problems with communication between the software. That being said, FMOD's Unity integration is very solid and highly functional. Before diving further into the comparison, it is useful to shortly run through FMOD's features as well.

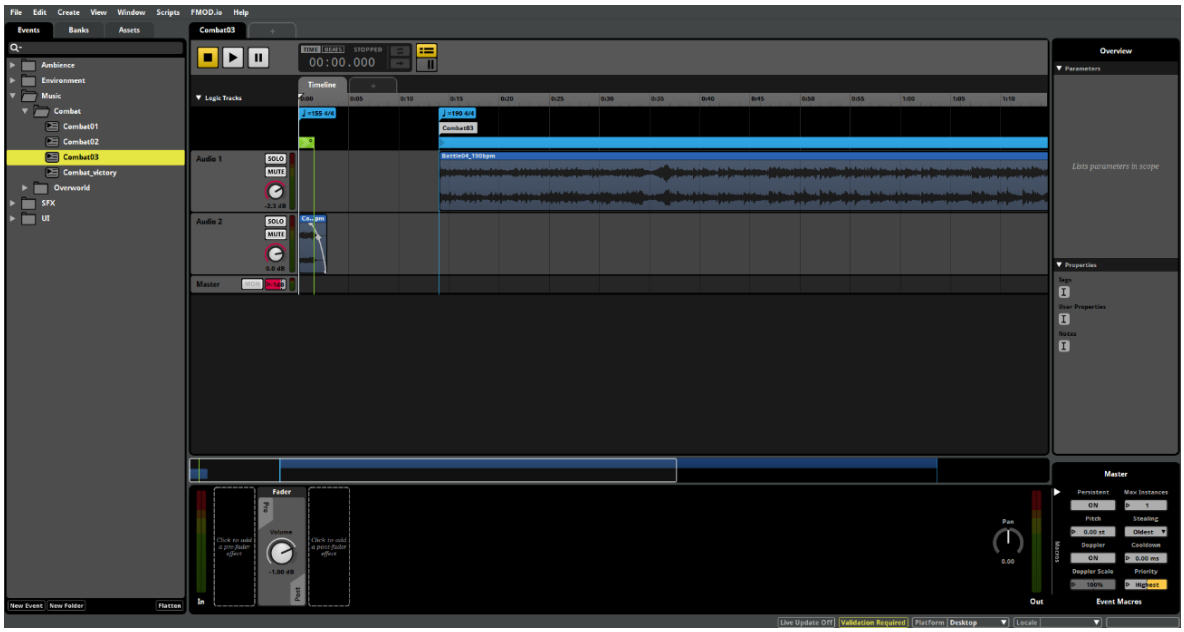
2.2.1 Events and audio banks

Like Unity, FMOD supports several audio formats like WAV, MP3, OGG, AIFF as well as console specific formats like VAG and many more (FMOD User Guide, 2019).

In FMOD audio clips are operated in events that are triggered in a game scene. Major difference to Unity is the Timeline window, which is very similar to a view in digital audio workstation. There can be one or several tracks in a single event, and they can contain several audio clips. Audio tracks and clips can be layered and played at the same time or individually but randomized. A number of parameters like volume, pitch, panning or effects parameters can be randomized as well on each instance of the event.

Event timeline is especially useful for adaptive sound effects and music – events that have several sections that evolve over time or by a trigger in the game. It allows to create adaptive music events for instance in a case where there's a stealth section in the game with music playing. When enemies are in alerted state the music become more tense with additional layers coming in. When combat is engaged the music might further evolve with more intensity and percussion stems

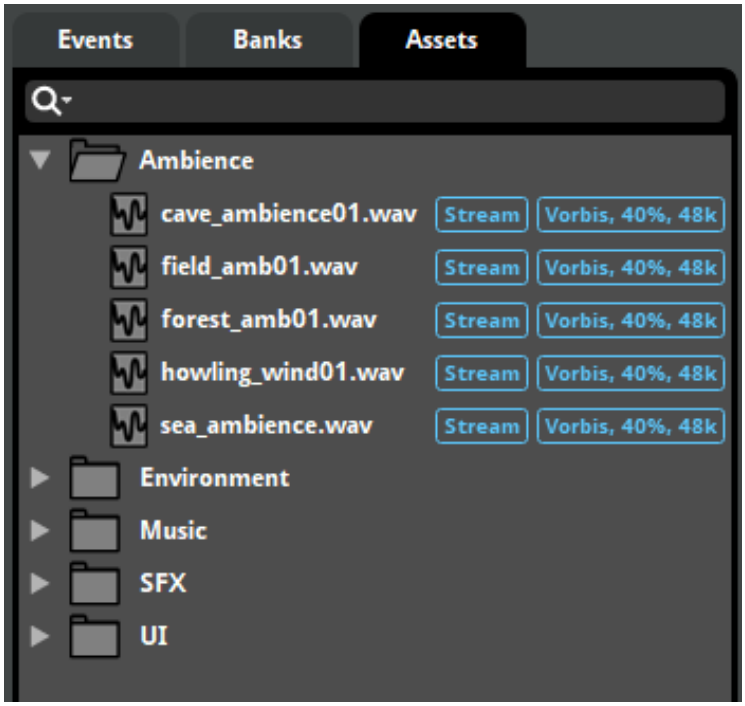
coming in. In FMOD timeline it is possible to add tempo, time signature and section markers which makes working with music particularly flexible. This kind of system adds a lot of depth and immersion into the soundscape of a game. The timeline works similar to an editor view in most digital audio workstations where it is possible to create tracks, edit audio clips, sequence and fade them together and add insert effects.



PICTURE 6. Event timeline in FMOD (Takanen A. 2020.)

Banks are collections of audio events that are compressed and loaded in the game when necessary. Only the banks that are used in a game scene are loaded at a particular time. One event can belong to several banks if needed. Banks are integral in optimizing the memory consumption of the game's audio system.

There are three load types for loading audio clips at runtime. *"Compressed"* means that the audio clips are compressed according to the compression amount determined in Encoder Settings. It is the most common load type for sound effects. *"Decompressed"* means that when a bank loads the files are decompressed into PCM data instead of being decoded as they are played. This is only recommended for situations where the CPU usage must be particularly low. *"Streaming"* means that instead of loading the entire audio clip assets are loaded into memory in parts as they are needed. This is great for long audio files such as ambience or music. (FMOD User Guide, 2019.)



PICTURE 7. Asset load type and quality settings in FMOD (Takanen A. 2020.)

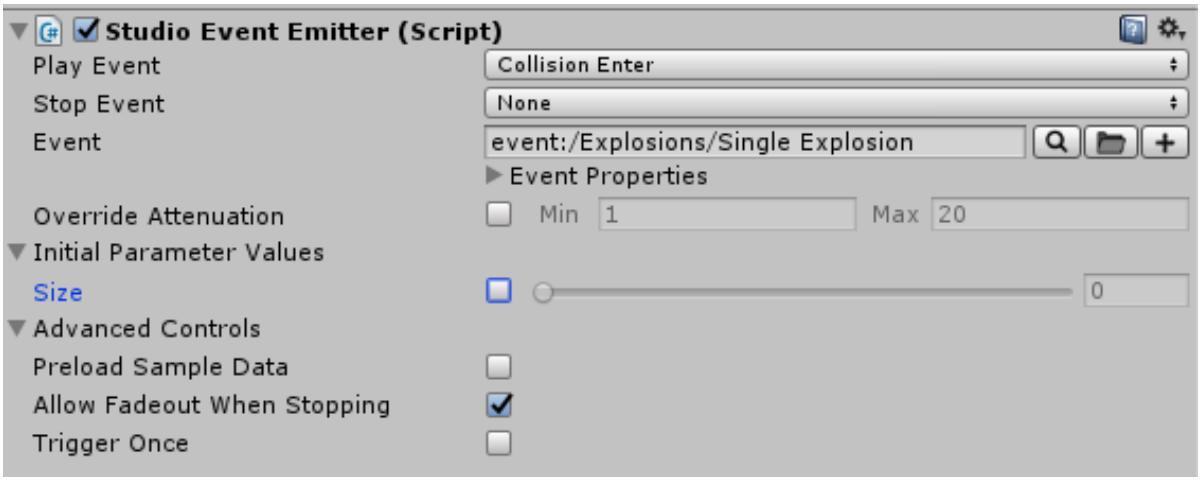
2.2.2 Studio Listener and Studio Event Emitter

Like Unity FMOD has its own audio listener component called Studio Listener, which is usually attached to the main camera. Studio Listener is required for 3D sounds to play as intended.

Studio Event Emitters are like Audio Sources in Unity both of which are used to trigger sounds in a game scene in Unity. However, where Unity's Audio Source also includes controls for the audio clip, Event Emitter only has the options for triggering an audio event. More in-depth audio controls are accessed in FMOD. To summarize, with FMOD audio event triggers are the some of the very few functions that need to be done in Unity. Sound design, mixing and hierarchy for the audio events is all done in FMOD.

Studio Event Emitter has options for not only playing but also stopping audio events. Event Emitters can also trigger FMOD Snapshots, which is useful for triggering specific mixer parameter states. Some of the options for triggering events are when a game object is loaded or destroyed, a game object is enabled or

disabled and entering a collision box. The options available cover many common situations, but if something more specific or contextual is required for triggering an event, some scripting needs to be done. FMOD's documentation for scripting events and Unity integrations should provide enough information to make different kinds of custom audio event triggers.



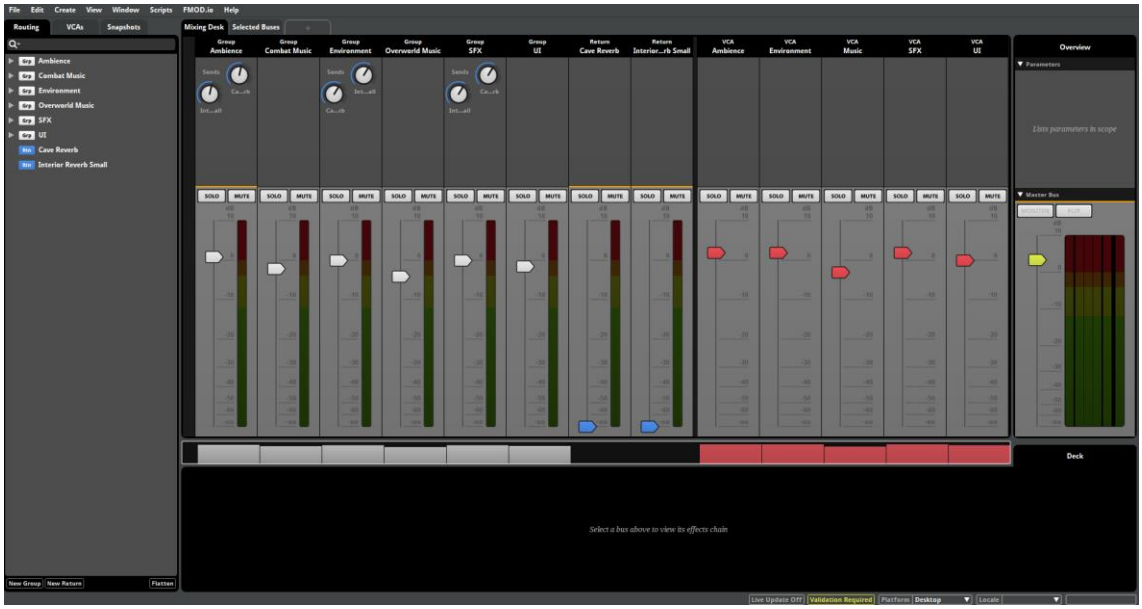
PICTURE 8. Studio Event Emitter component in Unity (Takanen A. 2020.)

2.2.3 FMOD's Mixer

FMOD's mixer view is divided into few sections. In the Routing tab events, groups and buses can be managed and routed. By default, events are routed to the master bus output. It is possible to create VCAs in the VCA tab which allows to control the volumes of different buses that do not share routing. (FMOD Documentation, 2019.)

Snapshots tab, like in Unity, is for capturing the state of mixer parameters which can be triggered and recalled in the game scene. Snapshots are particularly useful for creating reverb zones. Snapshots are triggered with Studio Event Emitters in Unity which allows to recall snapshots with different reverbs or other FX in different areas of the game. Unlike in Unity, triggering or transitioning between Snapshots does not require any scripting unless very specific custom behaviour is desired. A snapshot can either override the previous one or blend to one another.

Main part of the mixer window is the mixing desk. It is a view of all buses, auxiliaries and the master bus. All groups, return buses and unassigned events go through the master bus. Effects, EQ and compression can be applied to any of the tracks. The order of the applied effects modules can be altered simply by dragging them.



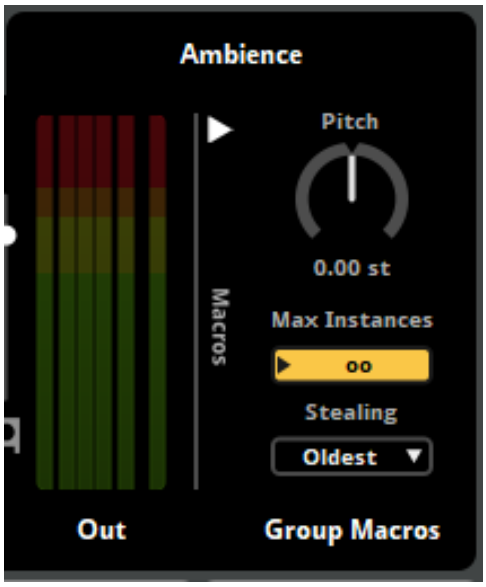
PICTURE 9. FMOD's Mixer window (Takanen A. 2020.)

It is possible to limit the number of audio instances passing through a group bus. In the Group Macros the maximum number of instances can be set, as well as stealing behaviour for excess instances can be selected. "*Oldest"* simply stops the oldest event instance replacing with the new one. *"Quietest"* stops the quietest event of the group mixer hierarchy and replaces it with the new instance. *"Furthest"* stops the furthest event instance from the listener regardless of its volume level. (FMOD Documentation, 2019.)

In addition to stealing, the concept of virtualization is important when it comes to limiting audible sounds in FMOD. A virtualized audio event is an event instance that is playing but not producing any audio. Once the event starts producing audio again, it becomes "real" instead of virtual. Virtualization always steals the quietest event instance. However, virtualization does not stop the audio track's playback – it simply mutes the master track. When an event stops being virtual and becomes real again the track becomes audible creating an illusion of a sound being in or out of hearing distance. (FMOD Documentation, 2019.)

It is also possible to limit the number of individual audio events from the event timeline's Event Macros section. Event instance limiting helps in keeping the overall soundscape clear and not overcrowded even in hectic sections of the game.

Similar to Unity, FMOD's mixer view can be customized to hide unwanted tracks and only show the desired objects or groups. Any groups, return buses or VCAs can be added to the mixer view. Several customized mixer views can be created – for instance character sounds, enemy sounds, ambience, UI sounds, music and effect return buses could have their own mixer views to make balancing volumes more convenient.



PICTURE 10. Bus instance limiting (Takanen A. 2020.)

2.2.4 Profiler & Performance

FMOD has a Profiler window which can be used to monitor and iterate the performance of the audio system. It is possible to monitor and record what the audio engine is doing while the game is played. (Robinson, 2019.) FMOD can be connected to Unity via the Live Update function so that Unity and FMOD are in sync real-time. This makes it possible to adjust and mix sounds live in the game scene and collect performance data in the Profiler.

Like with Unity's profiler, it is possible to collect data of the memory use, signal levels of Events or buses, the number of voices active as well as the number of instances of an Event being played and more (Robinson, 2019). FMOD's Profiler allows to get very specifically into the behaviour of individual Events or buses which is extremely useful for optimizing performance and trouble shooting. Especially the recording function makes it possible to examine the audio system's performance in detail.

Although when using FMOD integrated to Unity it is wise to examine both, Unity's profiler for overall performance of the game as well as FMOD's profiler to get detailed information about possible performance issues. While Unity does collect some data of the audio performance even when Unity's native audio engine is not in use, FMOD's profiler is needed for comprehensive performance monitoring and troubleshooting.

With the recording function it is possible to capture performance data of the gameplay. By creating new Profiler Sessions, the different gameplay recordings can be compared and determined whether performance optimization has been beneficial or not. The performance of specific tracks can be monitored by removing unwanted objects in a session. Events, groups and buses can also be shown if needed. Highly detailed data like the distance of 3D audio events in relation to the master bus or lifespan duration of individual events can be collected. (Robinson, 2019.)

Recorded profiler sessions can be played back whenever needed. There are two modes for the session playback. Waveform Playback Mode plays back the session as it was recorder. Making changes in the project does not affect the recorded session in this mode. Randomized audio events play exactly like they occurred in the recording session. Simulate Playback Mode calls the events from the session so randomized playback and parameter instances have different results like in the real gameplay scenario. Simulate Playback Mode is useful for debugging or testing adjustments in a real game scenario. Changes made in the project are updated real time. (FMOD Documentation, 2019.)

Profiler recordings can be named, managed and organized in folders. Sessions can also be exported and shared to other team members for further inspection or sent to FMOD support if necessary. In conclusion, FMOD offers extensive tools for detailed performance monitoring which are necessary for troubleshooting and optimizing a game’s audio system.

3 ANALYSIS AND COMPARISON

Now that the basic features of both audio engines are covered there is a fundamental basis to start comparing them. The analysis is divided into few sections: technical features, workflow, ease of use and resource investment. The comparison examines both audio engines' technical and artistic sound design potential, different aspects of their respective audio implementation pipelines and how they might affect the audio work and rest of the development team.

3.1 Comparison of technical features

Both audio engines provide the sound designer with similar set of basic tools. Both Unity and FMOD make it possible to import clips, compress them, trigger audio events, create rules for them, add effects, mix and build hierarchies as well as monitor the sound system's performance. As far as the sheer amount and depth of technical features go, however, FMOD provides more in-depth functions and flexibility. On the other hand, many options that Unity lacks can be worked around with good planning and custom scripting. Overall, both engines have the obvious requirements for creating a fully functional sound system for a game.

One of the biggest advantages of FMOD is the event editor – especially the timeline view. It gives freedom to edit and manipulate sound assets to great extent even after importing them into the audio engine. It is also possible to construct SFX out of individual layers put together in the audio engine. For example, instead of using variations of a full gun shot SFX and randomizing between them it is possible to import variations of individual layers of the gun shot sound: mechanical layer, body layer, sub layer and tail layer. These layers are then put together and mixed in the audio engine itself which gives more flexibility to fine tune the complete sound effect and make it work in the game's context. Constructing complex and important sound effects from layers in the audio engine is a great way to add more variation to each instance of the SFX. Instead of randomizing between only few audio clips, each individual layer of the sound and their parameters are randomized resulting to significantly more variation between

each event instance. This entire process can be done with FMOD's built-in tools without any additional programming, whereas in Unity even the most basic audio clip randomization must be done via code.

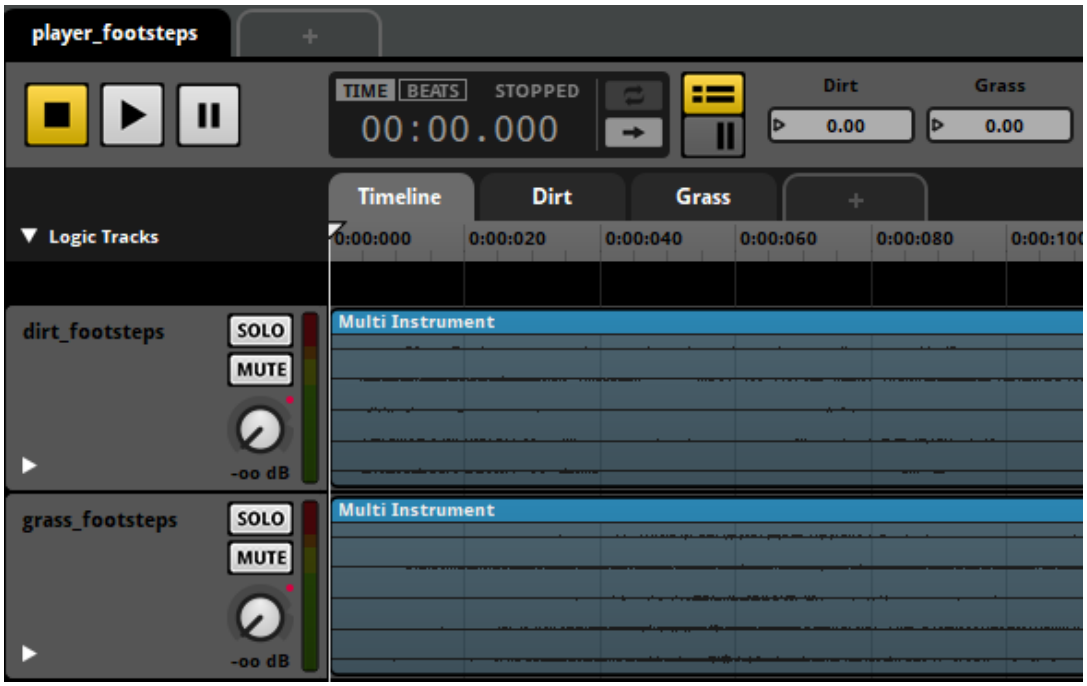
FMOD's timeline also makes it possible to sequence music and sound effects from one section to another, which is not easy to accomplish in Unity without using several Audio Sources and programming to trigger and stop them whenever needed. In fact, triggering audio any other way or any other time than on the load of the game object requires scripting. When working with Unity it is beneficial for the sound designer to know basic audio event scripting to cover some of the most common scenarios where programming is needed for audio work. Overall, compared to FMOD's features Unity's audio clip editing and audio triggering capabilities are lacking. It requires more work and custom programming solutions to achieve some of the effects and behaviour that is already built in to FMOD. Fortunately, Unity's documentation does cover the basics of audio related scripting quite well.

Mixing, routing and FX capabilities of the two engines are seemingly similar. Both engines have a mixer, buses and a selection of audio effects to work with. Both engines also allow customization of the mixer views. FMOD provides some more mixing flexibility with the inclusion of VCAs. Major difference, however, is FMOD's event and group macro controls, which allow event, group and bus instance limiting. In addition, FMOD has the powerful Parameter feature which can be used for great number of purposes from dynamic sound and music to mixing to real-time effects parameter automation.

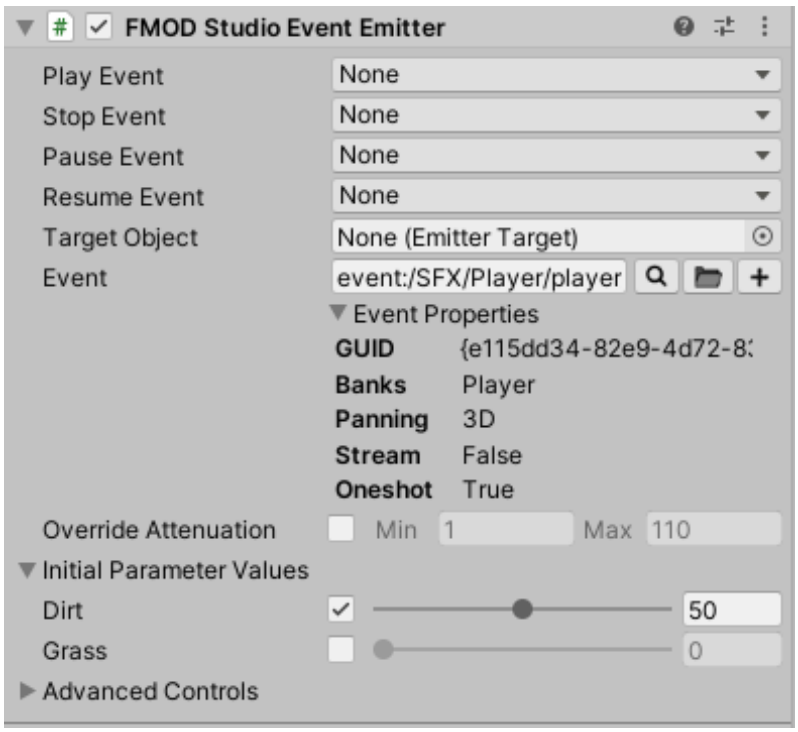
Unity only features a global audio voice limit, which determines how many audio voices can play simultaneously in a game scene (Fisher, 2020). Audio Source Priority is used to determine which sounds are muted when the voice limit is reached. The Priority slider has range of values from 0 to 256 where 0 is the most important and 256 is the least important. An arbitrary value such as this is somewhat difficult to work with especially in projects with large number of audio sources and voices. In Unity the only way to create an audio hierarchy is to go through every single Audio Source and keep track of the source priority values. In contrast, FMOD a maximum instance limit can be determined for each event

or group separately. A clear number per event or group is significantly easier to work with compared to Unity's vague 0 to 256 value. Additionally, in FMOD it is possible to work on hierarchies within and between SFX groups by limiting the number of voices that go through buses, as well as get into details by limiting instances of individual events. Overall, FMOD's audio voice limiting system is more comprehensive and flexible.

Another significant technical advantage FMOD has over Unity is the parameter control and automation feature. With custom made FMOD parameters it is possible automate volume levels or FX parameters real-time in the game. FMOD's parameters are useful for creating dynamic audio systems like footstep sounds, which change according to the material player is walking on or dynamic music system where FMOD parameter controls the music intensity. Parameters are some of FMOD's most powerful features, which provide almost endless possibilities when it comes to creating adaptive audio systems. The real time parameter control as a feature and a concept that might take some adjustment when starting to explore its potential.



PICTURE 11. Parameter controls for footstep materials in FMOD (Takanen A. 2020.)



PICTURE 12. FMOD's footstep SFX material parameters in Unity's Studio Event Emitter component (Takanen A. 2020.)

When properly implemented real time automation can add breath and realism to the game's sound design making the player's actions dynamically affect the soundscape. These types of features are completely absent from Unity's audio tools. Interactive sound implementation, such as parameter automation must be solved and executed in the code. Many of the tools and functions that Unity is unfortunately still missing are very much the ones that make a game sound system dynamic and alive. Unity does have the Snapshot feature, which makes it possible to do some dynamic sound implementation, but triggering and transitioning between Snapshots does often require scripting. That goes for most real-time sound and effects parameter manipulation as well.

While Unity's profiler is a capable tool for gathering data of the game's performance and troubleshooting, for purely audio purposes FMOD's profiler provides more in-depth information of the audio engines behaviour. FMOD's profiler's recording function makes it particularly useful for troubleshooting and optimizing specific sections of the game. When optimizing the sound system, FMOD's profiler is a better tool. However, with FMOD integrated Unity's profiler collects data of FMOD's audio performance as well. It is useful when examining the game's overall performance and memory use in Unity.

FMOD's audio features are in many aspects more advanced compared to Unity. However, one advantage of working with Unity's audio engine is the opportunity to work inside the game engine itself. Working with two separate software is always more sensitive to integration related issues. Nevertheless, no issues with the integration have occurred with FMOD and Unity versions used in the study.

3.2 Workflow

For the workflow analysis some of the topics discussed are audio pipeline's efficiency and flexibility as well as time use of the sound designer and the rest of the development team.

Positive aspects of using Unity's sound engine is the simplicity of working in a single development platform with no need to run any parallel software. Many developers are very familiar and comfortable with Unity, which could make it their preferred choice for audio implementation as well. For developers with less knowledge about sound but vast experience on Unity, working with its built-in audio engine might be a viable choice. Extending and customizing Unity's audio features via scripting when necessary might not be an obstacle to achieve the desired results. In some cases, the advantages of using a familiar working environment might overweight FMOD's more extensive built-in features. Some projects, especially on development platforms with significant hardware limitations, might not have use for most of the advanced tools that FMOD provides. Nevertheless, some commonly used sound design features like parameter and clip randomization would be a necessary addition to Unity's set of audio tools in future updates.

However, for sound designers who are familiar working with audio FMOD is better equipped with tools for building a high-performance audio system. Features like clip editing, timeline, parameter randomization, real-time parameter control, layering, effects, mixing and profiling offer more than enough tools for sound designer to work with minimum programming required. Sound designers often have

little to no experience on programming, so having as much of the required functionality available as possible improves and streamlines the workflow significantly. The sound designer is able to work efficiently even without programming skill, and without programmers having to spend valuable time on solving audio related problems or creating custom tools for the sound system. The sound designer is able to utilize his or her expertise to the highest capacity, and the vision is less likely to be compromised due to miscommunication with other members of the team. When programmers implement audio related functionality, it is extremely important for the sound designer to clearly communicate what it is that is required. The sound designer has a trained ear and an expert understanding of what the sound should do and how it should behave. The less the sound designer is able to be involved in the audio implementation first-hand, the more the overall sound design is going to suffer.

FMOD's timeline editor and interface similar to many digital audio workstations is more likely to appeal audio engineers than Unity's significantly less visual audio editor. FMOD makes it possible to edit, manipulate, layer and mix audio clips in the audio engine itself which significantly reduces the need to go back and forth between DAW and implementation software. It allows the sound designer to audition and tweak assets real time in the game's context. The closest feature to FMOD's timeline editor that Unity has to offer is incidentally called Timeline which is meant provide visual tools for creating cinematic content and audio-visual sequences. The Timeline tool can be downloaded and installed as a separate Unity Package. The Timeline is used for working with animation, visual effects as well as sound effects. It is a useful and visual tool for creating cut scenes and gameplay sequences with tightly synchronised sound and animation. (Unity Guide, 2020.)

One of FMOD's greatest advantages is for sound designers to be able to implement audio and create a sound system with minimal assistance from programmers. That is not to say that FMOD cannot be customized with code and integrated into all kinds of game projects. The engine and its audio behaviour are highly editable in C# or C++. In fact, FMOD Studio Unity integration exposes the full API in C# or C++ for programmers to fully operate and customize the engine

according to the requirements of the project at hand. (FMOD Documentation, 2019.) The user guide covers FMOD's C++ and C# functionality quite extensively.

FMOD has additional Reaper integration, which can streamline the workflow of sound designers who use the Reaper digital audio workstation for audio asset creation. It is possible to link a Reaper project into FMOD. All rendered files from a linked Reaper project are imported as linked audio assets in FMOD. Additionally, it is possible to import a full Reaper timeline. FMOD's Reaper integration helps especially with audio asset management. Asset naming, time selection of the rendered clips and rendering file path can all be automated which makes continuous rendering and management of asset iterations considerably faster (FMOD Documentation, 2019). It is even possible to render a complete timeline of a music track as stems straight into an FMOD audio event. This can be particularly useful for working with dynamic music systems.

FMOD has great working prerequisites for studio teams or outsourced audio that work remotely. The FMOD sessions are separate from the game project itself, and the audio banks make it possible to transfer and share assets in the version control compressed without the need to download large audio files. Also, recorded profiler sessions can be shared to other members of the team for detailed troubleshooting. There are benefits in being able to separate the audio work from the main game project.

3.3 Ease of use

When examining the ease of use of both audio engines, the analysis is not only about software design or the user interfaces, but especially about the skill requirements and intuitiveness of operating each of them respectively.

Unity has been and continues to be designed for developers of variety of expertise in several different aspects of game development. It is a major design challenge to provide the tools for working in all these different areas of development that are not only competent but also presented in a cohesive, user friendly interface. There is, however, certain consistency in Unity's editor, user interface and

different components which eases developers' learning curve when it comes to new or unfamiliar features. Recent and upcoming updates are focusing on improving the user experience further via user interface and workflow improvements while still maintaining familiarity for long time users (Kahn, 2019.) This cohesiveness and familiarity of Unity's editor can be of significant assistance to many when working on audio even with limited sound implementation experience. Furthermore, the ability to work inside the Unity game engine itself certainly adds to the approachability and ease of use of the audio engine.

Feature-wise Unity's audio interface is more stripped down and, in some respects, even lacking compared to FMOD. However, the absence of several features also comes with simplicity of operating it. However, complications arise when the lacking features of Unity's audio engine are needed. One clear shortcoming in the user interface of Unity's Audio Source component among others, is the lack of units in many of the adjustable parameters, which should indicate what exactly is being modified. It is unclear if the values adjusted are decibels, percentages or just arbitrary numbers designed for modifying the parameters. For developers who are less familiar with the units used in audio engineering this might make operating the interface more straightforward. However, for experienced users who come from the field of audio engineering or sound design this can be hindering and frustrating. Fortunately, the Audio Mixer view is better equipped with units and indicators for each adjustable parameter. It is unclear why the parameters in Audio Source components don't have units in them – it could be a remain from an old Unity feature that is hopefully revised in future releases. Unity's audio engine's user experience could be summarized as relatively simple to use due to the limited number of features, but perhaps unnecessarily complex once one starts to miss and subsequently work around those lacking functions.

FMOD, on the other hand, is significantly more packed with features. However, user interface is designed and laid out in a very simple, approachable manner. The interface is divided into distinctively separate tabs: Events, Banks and Assets, with editor timeline in the middle. In comparison to Unity and also other audio middleware FMOD's visual presentation is arguably one of the most user-friendly designs. Despite the relatively simplistic layout there is more going on in

terms of audio implementation functionality which also adds to the learning curve of the software. From sound design perspective FMOD has more potential, but naturally it comes with more study and consideration for functions that are completely absent from Unity's user interface. For developers who are inexperienced in the field of audio implementation FMOD can seem slightly intimidating at first compared to Unity which to great extent is due to the significantly higher number of features it has. However, for sound designers who expect the inclusion of these features in an audio middleware FMOD as a software is very user-friendly in its design. There is a lot of empty space in the interface which makes the editor easy to navigate.

Of course, one aspect of using a dedicated audio middleware is having to operate two software simultaneously. Most of the sound design work happens in FMOD, but for example operation of the audio event triggers is one of the few things needed to be done in Unity. In the beginning of a project FMOD must be integrated into Unity with consideration for certain issues like what files are included in version control. There are few ways to work FMOD into the version control, which along with the whole Unity integration process are described step-by-step in FMOD's documentation (FMOD Documentation, 2019.)

Overall, FMOD is very approachable in its visual design. The learning curve of the software is not created by its user interface but by the quantity of its implementation functions. The extent of FMOD's features as well as its nature as third-party software integrated into Unity might require more involvement from the sound designer early on. Sound design expertise is required with FMOD more so than with Unity, although basic audio implementation with FMOD is possible even with limited audio engineering knowledge. FMOD certainly is the easier-to-use solution for projects that have high requirements for the game's sound system, and which require the advanced audio implementation features.

3.4 Resource investment

Comparison of the two audio engines can be broken down to two fundamental aspects to which they affect in the development process: the quality and flexibility

of the sound system and resource investments required to produce it. As far as technical features go, FMOD has better tools and greater potential for creating a game audio system. From workflow perspective, FMOD has its advantages by allowing to do major portion of the sound design in the audio engine's editor itself while minimizing going back and forth between digital audio workstation and the audio implementation software.

When discussing resources in game development mostly time, working hours, and financial costs are addressed. The nature of the game project, the scope of it, the budget and the development team with all the skill sets available are all in complex interaction with one another. The project dictates what is needed in design, code, art and audio departments, and the budget and schedule put certain constraints on how and how quickly everything should be produced. Because of all the variables in play there is no single correct solution for audio implementation. However, some general level conclusions can be made about the resource investments of sound implementation in Unity and FMOD audio engines.

Time management depends heavily on the structure, size and skill set of the development team. The analysis is done mostly from the perspective of small to middle sized development team. Especially because of the recent shift in the game industry, and success of indie games over the last few years produced by very small teams it is reasonable to analyse the situation from a scenario where resources are fairly limited.

Having at least one dedicated sound designer as part of the team is quite common, but there are developers especially in the mobile game market who choose to invest in other aspects of the game and implement the audio themselves to the best of their abilities. In a scenario with no specialized sound designer in the team Unity's audio engine might be a viable choice since Unity as a working environment is familiar to many, and from audio perspective it is more stripped down. This could be the situation, for example, in a mobile game company with only two or three employees. It could be viable, and from resource point-of-view efficient to implement audio yourself and possibly commission at least some of the assets. In mobile and browser games the core loop is much more based on immediate action and reward cycle than on story or sensory immersion (Wolstenholme,

2017). Hence, the emphasis of the sound design should be mostly in feedback – the sound must add to the pleasant and rewarding feel of the core gameplay.

However, games that in addition to gameplay appeal with their story, world and audio-visual presentation have higher technical standards to reach with sound design and sound implementation. In these projects a dedicated sound designer and proper audio implementation tools are crucial. The more work there is to be done in sound implementation, the more time it would take to customize and add to Unity's audio engine's shortcomings. In technically demanding projects FMOD's comparatively minimal programming requirements save a lot of sound designers' as well as programmers' efforts allowing them to spend their time more efficiently.

Nevertheless, operating FMOD does not come entirely without the need for programming. The difference to Unity, however, is that FMOD's tools or functionality does not require customization via code to achieve commonly used sound implementation techniques like parameter randomization. When implementing audio with FMOD the most common situations where additional programming could be required are when playing, stopping or pausing audio events in very specific or unusual scenarios. It is possible to customize FMOD's audio implementation capabilities via code, and for a fee even get access the source code for in-depth tailoring. This could be required for demanding and large productions where audio-technical requirements are particularly high or specific.

An aspect to consider when it comes to financial expenses is FMOD's license fee. There are several licenses to choose from all of which correspond to the development budget of a game project. The licence gives permission to distribute FMOD in a game. For projects with a development budget under $500k the lowest level license named "Indie" license is free for a single game. For development budgets between $500k and $1.5M the "Basic" level license per game is $5,000. The highest level "Premium" license for projects with development budgets over $1.5M costs $15,000 per game. (FMOD Licencing, 2020.) The license per game even in the highest level is relatively inexpensive considering the development budgets they correspond with.

Furthermore, there are optional features that can be purchased in addition to the licence. One year of dedicated email support service can be purchased for $5000 for any level of the licence. For projects with the highest Premium level license source code access is available for $15,000 which could be necessary for particularly large or demanding projects, which require unique customization in the audio technical department. (FMOD Licencing, 2020.) In conclusion, the financial investment in the licence itself can range from nothing at all for projects with modes budgets to tens of thousands of dollars for demanding high budget projects.

4 DISCUSSION

In any game project choosing the best tools and system pipelines is extremely important. The decisions made early on have far-reaching and long-lasting consequences, and it might be increasingly difficult to change the course as the project advances. The significance of the decisions in the early stages of a production, as well as the surprising amount of developers who are unfamiliar with third-party audio tools and their potential benefits introduces a demand for this kind of analysis. The more informed developers are about the tools available, the better they can maximize the quality of the production with realistic investments. One purpose of the study is to provide a resource that helps in making a well-informed decision regarding the audio engines for developers who are considering the options. It should be stressed that there is no objective correct solution that applies to every single project since the applicability and potential of the audio engine depends on the skill set, budget, production time and technical requirements of the development team.

Based on the study there are certain conclusions that can be made about Unity's audio engine and FMOD. The two audio engines have similar basic sound implementation functionality for compressing, loading and playing audio clips, mixing, creating hierarchies and monitoring the audio system. Unity offers less options for shaping and controlling single audio clips as well as groups of sounds. Unity also tends to be less precise with many of the parameters, which might add to the ease of use for those who are less familiar with in-depth audio work, but overall is a considerable shortcoming of the software. FMOD does offer more control over every aspect of the sound implementation process, but despite the easy to use interface it does require more knowledge about game sound design concepts and audio engineering to make use of many of its features. On the other hand, sound implementation in Unity requires more programming expertise and scripting knowledge, which for many sound designers might be a major challenge.

Most of the analysis is done from the perspective of a single project scenario. However, there is something to be said about the long-term potential and effect the audio engines have for production quality and efficiency. Compared to FMOD

majority of the advantages of using Unity's audio engine are relatively short-term gains. Most of the upsides of Unity are about not having to adjust to a new workflow, learn to use a more elaborate, feature heavy audio implementation software and deal with integration related challenges. Overall, the main reason to not switch to FMOD, the objectively more capable audio engine, is in a case where the change introduces more challenges than benefits. If adjusting to the new audio implementation pipeline takes more working hours than it eventually saves, or if the improvements in the production quality are not evident enough perhaps due to the learning curve of the software, the switch might not feel justifiable. Of course, it is impossible to accurately predict how exactly decisions like this affect a production, and if the consequences are overall in the positive side.

As discussed, the benefits of using Unity's audio tools almost completely arise from the risks that switch to FMOD might bring especially for developers with limited audio expertise. Compared to FMOD there are very few, if any technical advantages in unity's audio system itself. FMOD might not be the better option for every single development team from the perspective of a single project, but as a more far-sighted solution it could be considered the superior audio engine.

As a long-term objective, if it is possible for the team to invest some time and resources to learn and polish the workflow with FMOD it will pay off as increase in the quality of the game's sound system and saved working hours especially for programmers. This will ultimately help a game development team to grow and refine as a unit improving their overall production quality and efficiency. The time invested in one project for learning the better tools might very well pay off by the next production. This is why making the switch to FMOD is possibly the better long-term choice if the team is capable of handling the adjustment timewise and financially.

In Unity's case the challenge of providing a full-fledged game development platform that is able to completely stand on its own is certainly ambitious. Audio implementation is only one of the many aspects of the software, and it is understandably not the top priority in the list areas to improve. Perhaps some of the Unity's lacking audio functions are added, and some of the existing features are

polished in future updates. However, as of 2020 there seem to be no official statements about major audio related improvements in upcoming updates. The focus of Unity's 2020 updates seems to be in stability, workflow and quality of life improvements which could more or less affect audio work as well (Unity Blog, 2020). Unity Technologies is of course very aware of third-party audio engines and understands that some of them offer more advanced and capable audio implementation tools. It is only logical for Unity to embrace and support the use of third-party tools for covering some of the still inadequate features of the game engine. After all, the main ambition for Unity Technologies is to offer the most well-rounded, reliable and approachable game engine in the market. It means that the attention in updating the software needs to be focused foremost on certain broad aspects like user experience and workflow more than specialized areas like audio implementation. It makes sense for Unity to support the integration of third-party solutions and keep working on the big picture.

Third-party audio solutions introduce the opportunity of outsourcing sound design services to other companies. Alternatively, adapting these tools into a company's repertoire makes it possible to buy services from outsourcer companies. The benefit of collaborating with an audio outsource company is that it is an all-in-one service that covers all sound and music related needs, which could be significantly more expensive to commission from separate sources or too challenging and time consuming to produce in-house (MCV Develop. 2016). An outsourcer company can provide high level specialization and production facilities while the game studio as employer remains in control.

On the other hand, many game companies that focus on developing their technical and artistic in-house capabilities also end up providing outsourcing services themselves expanding their business potential significantly. Gaining experience on working with different platforms, tools and software can make a game studio's in-house talent a very valuable service. This kind of long-term trajectory of working methods and the development team as a whole might be easily overlooked in perspective to more short-term project to project way of thinking. However, in the game industry this kind of long-term development of the company through the growth of the individual seems to be more and more adopted mentality. Some companies actively reject the traditional senior and junior developer hierarchy

and choose to invest in the skills and especially the potential of the individual. This kind of slow but perceivable shift into a more people oriented direction is very welcome in an industry where unrealistically tight schedules and overworking to often unhealthy extent are unfortunately very common.

REFERENCES

Audiokinetic. Pricing. Read 12.2.2020.
https://www.audiokinetic.com/pricing/

Dillet, R. 2018. Unity CEO Says Half of All Games Are Built on Unity.
https://techcrunch.com/2018/09/05/unity-ceo-says-half-of-all-games-are-built-on-unity/

Fisher. J. Audio Tutorial for Unity. Read 15.3.2020.
https://www.raywenderlich.com/532-audio-tutorial-for-unity-the-audio-mixer#toc-anchor-006

Hocking, J. 2018. Unity in Action, Second Edition: Multiplatform Game development in C#. Manning Publications.

FMOD. Licensing. Read 6.4.2020.
https://www.fmod.com/licensing

FMOD. User Guide. Read 18.3.2020.
https://www.fmod.com/resources/documentation-unity?version=2.0&page=user-guide.html

Kahn, D. 2019. Evolving the Unity Editor UX. Read 7.4.2020.
https://blogs.unity3d.com/2019/08/29/evolving-the-unity-editor-ux/

MCV Develop. 2016. Lend Us Your Ears: The Benefits of Audio Outsourcing.
https://www.mcvuk.com/development-news/lend-us-your-ears-the-benefits-of-audio-outsourcing/

Philipp M. 2015. How to Use Audio Mixers in Unity 3D. Read 15.4.2020.
https://www.studica.com/blog/how-to-use-audio-mixers-in-unity-guide

Robinson, C. 2019. Game Audio with FMOD and Unity. Routledge.

Unity. Unity Blog. Read 20.4.2020.
https://blogs.unity3d.com/2020/03/19/state-of-unity-2020-in-this-together/

Unity. Unity User Manual. Read 17.2.2020.

Wolstenholme, K. 2017. What Is a Core Loop in a Mobile Game? Read 6.4.2020.
https://risinghighacademy.com/what-is-a-core-loop-in-a-mobile-game